Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut ohjelmistot

2012

Stefan Roos

ÄÄNIMOOTTORIN SUUNNITTELU JA TOTEUTUS

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut ohjelmistot

Toukokuu 2012 | 48 s.

Ohjaaja: TkL Jari-Pekka Paalassalo

Stefan Roos

ÄÄNIMOOTTORIN SUUNNITTELU JA TOTEUTUS

Tässä opinnäytetyössä suunniteltiin ja toteutettiin DirectX:n XAudio2-kirjastoon perustuva äänimoottori. Tämän äänimoottorin tärkeimmät ominaisuudet ovat useiden äänien yhtäaikainen toistaminen, kyky toistaa OGG Vorbis –tiedostoja, äänen tulosuunnan määrittäminen 3D-avaruudessa sekä helppokäyttöinen ohjelmointirajapinta.

Työn suunnittelua varten tutkittiin ja vertailtiin erilaisia äänimoottoreita, joista saatiin tärkeää informaatiota äänimoottorin toteuttamista varten. Lisäksi tutustuttiin WAV-tiedostojen sekä OGG Vorbis –tiedostojen rakenteeseen, jotta niitä voidaan äänimoottorilla toistaa.

Äänimoottorin toteutettiin dynaamisena linkkikirjastona C++-ohjelmointikielellä, jolloin se on yksinkertaista ohjelmointirajapinnan avulla liittää sitä tarvitseviin sovelluksiin. Toteutuksessa käytettiin useita erilaisia olio-ohjelmoinnin suunnittelumalleja.

Projektin lopputuloksena saatiin Windows-ympäristöön kehitettävien pelien ja ohjelmien käyttöön soveltuva äänimoottori, jonka käyttö on yksinkertaista sekä toteutus on modulaarinen ja helposti laajennettavissa.

ASIASANAT:

DirectX, OGG, ohjelmointi, WAV, XAudio2

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology | Embedded Software

2012 | 48 p.

Instructor: Jari-Pekka Paalassalo, Lic. Sc. (Tech.), Principal Lecturer

Stefan Roos

DESIGN AND IMPLEMENTATION OF AN AUDIO ENGINE

The subject of this thesis was to design and implement an audio engine based on DirectX's XAudio2 library. The most important features of this audio engine would be the capability to play multiple sounds at once, change the position of the sound in three-dimensional space and an easy to use application programming interface.

A variety of different audio engines were studied and compared before the design and implementation, which gave a great amount of useful information. In addition WAV and OGG Vorbis files were studied so that the audio engine could support them.

The audio engine was implemented as a dynamic link-library using C++ programming language, which allows it to be easily used in programs that need it. Many different design patterns of object oriented programming were used in the implementation.

The result of the project was an audio engine for games and programs in Windows environment, which has modular implementation, is easy to use and can be easily extended.

KEYWORDS:

DirectX, OGG, Programming, WAV, XAudio2

SISÄLTÖ

KUVAT

TAULUKOT

KOODIESIMERKIT

SANASTO

DirectX	Microsoftin kehittämä ohjelmointirajapinta, jota voidaan mm. hyödyntää äänien, grafiikan ja erilaisten laitteiden hallintaan.
DLL	Windows-käyttöjärjestelmässä käytettävä dynaaminen linkkikirjasto.
OGG	Xiph.org yhdistyksen ylläpitämä avoimen standardin multimediasäiliötiedostomuoto
PCM	Pulssikoodimodulaatio.
Rengaspuskuri	Virtautuksessa käytettävä puskuri, johon kirjoitetaan ja josta luetaan samaan aikaan. Puskurin loppuun saavuttaessa siirrytään takaisin sen alkuun.
Virtautus	Menetelmä, jonka avulla tiedostoja voidaan lukea pala kerrallaan.
WAV	Microsoftin ja IBM:n kehittämä tiedostomuoto digitaalisen äänen tallennusta varten.
X3DAudio	XAudio2:n apukirjasto 3D-äänien luontiin. Kirjaston avulla voidaan laskea tarvittavat muutokset mm. äänenvoimakkuudelle riippuen äänen sijainnista.
XAPOFX	XAudio2:n apukirjasto, jota käytetään ääniefektien lisäämiseen. Kirjastossa on tarvittavat rakenteet efektien luontia varten sekä useita valmiita efektejä.
XAudio2	Microsoftin kehittämän DirectX:ään kuuluva ohjelmointikirjasto, joka on tarkoitettu äänien ja äänimoottoreiden ohjelmointiin.

1 JOHDANTO

Äänet ovat peleissä olleet aikaisemmin melko vähäpätöisessä roolissa, niiden toteuttamista on tyypillisesti alettu miettimään vasta sitten kun peli on jo muuten lähes valmis. Tämä on usein ollut todella haitallista hyvän äänimaailman luomisessa, koska peliin on vain nopeasti laitettu jonkinlaiset äänet. Nykyään peliäänet ovat iso osa pelikokonaisuutta, ja niiden toteuttamiseen käytetään huomattavasti enemmän aikaa. [1]

Äänet ja musiikit tarvitsevat kuitenkin pelissä jonkinlaisen järjestelmän, joka mahdollistaa sujuvan äänentoiston. Tässä tilanteessa nykyaikaiset pelit käyttävät äänimoottoria. Äänimoottori on yksi osa pelimoottoria, se hoitaa pelissä tarvittavien äänien toistamisen ja hallinnoimisen. Erillinen äänimoottori myös mahdollistaa äänten yksinkertaisen liittämisen peliin.

Tässä opinnäytetyössä keskitytään nimenomaan äänimoottorin suunnittelemiseen ja siihen miten sen toteuttaminen käytännössä tapahtuu. Tämä toteutettava äänimoottori tulee käyttöön pelinkehitystiimille, jossa olen mukana. Tämä pelinkehitystiimi on nimeltään FantasyCraft, ja se koostuu tällä hetkellä neljästä henkilöstä. FantasyCraft on koottu vuonna 2010, ja se on perehtynyt fantasia-aiheisiin peleihin, erityisesti roolipeleihin.

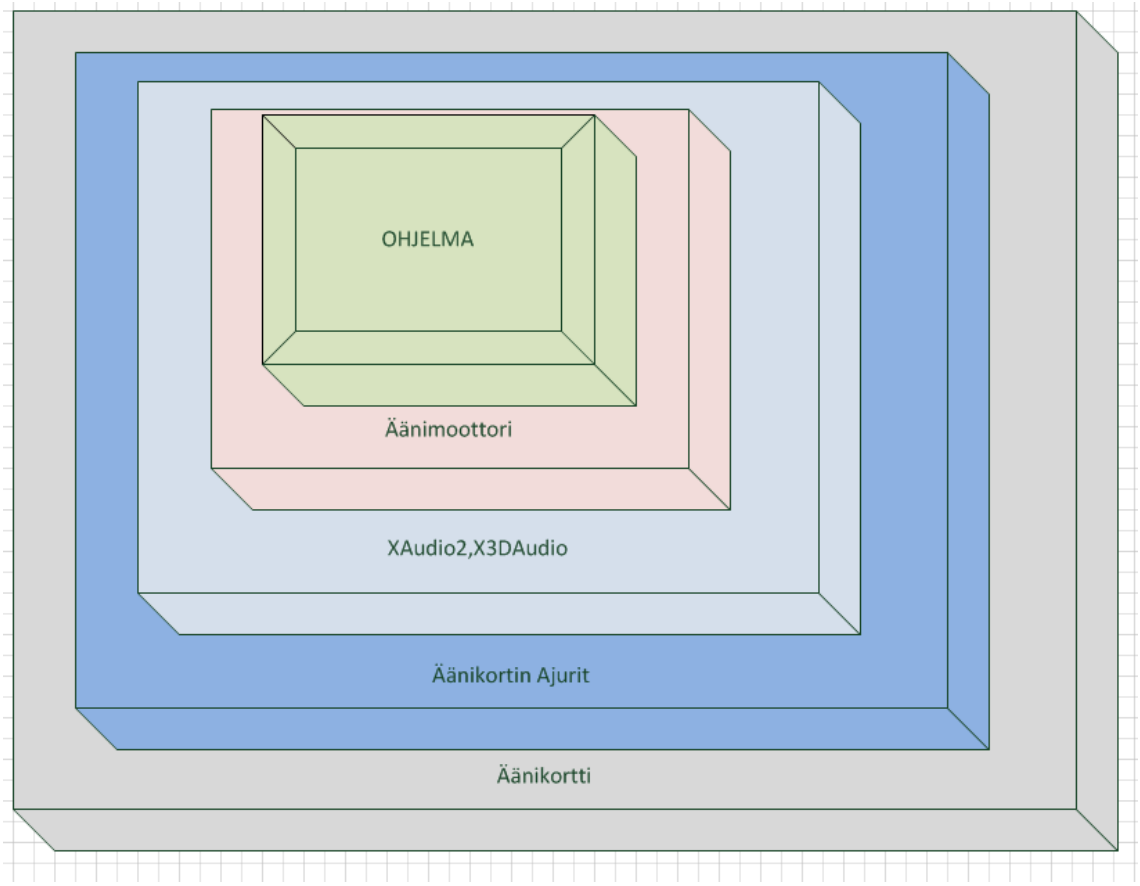
2 ÄÄNIMOOTTORI

Äänimoottori on se osa ohjelmaa, joka hoitaa kaiken äänentoistamisen. Äänimoottori on siis pelimaailman äänien ydin. Äänimoottoreita käytetään myös erilaisissa hyötyohjelmissa, mutta se on hiukan harvinaisempaa, koska useimmat hyötyohjelmat eivät tarvitse kovinkaan hyvää tai monipuolista äänentoistokykyä.

Äänimoottori mahdollistaa äänten käyttämisen ohjelmissa ja peleissä huomattavasti yksinkertaisemmin kuin kirjoittamalla erikseen jokaiselle ohjelmalle oma äänentoistotapansa. Äänimoottori on komponentti, jota voi käyttää yksinkertaisesti monissa eri projekteissa ja joka tarjoaa kaikki äänentoistamiseen ja joskus jopa äänen muokkaamiseen tarvittavat toiminnallisuudet.

Äänimoottorin tärkeimpiin toimintoihin kuuluvat äänitehosteiden ja musiikin toistamisen lisäksi mm. äänitiedostojen lukeminen, kompressoitujen äänien purkaminen sekä äänen saattaminen äänilaitteelle. Näiden toimintojen lisäksi äänimoottori voi muokata toistettavia ääniä lisäämällä niihin erilaisia efektejä tai muuttamalla niiden ominaisuuksia jollain muulla tavalla.

Kuvassa 1 on kaavio, josta selviää, missä tasossa äänimoottori toimii. Ylimpänä kuvassa on ohjelma, joka käyttää äänimoottoria. Tämän jälkeen on itse äänimoottori. Sitten äänimoottorin käyttämät kirjastot, jotka ovat osa DirectX:ää, äänikortin ajurit ja viimeisenä itse äänikortti.



Kuva 1. Äänimoottorin sijainti järjestelmässä.

2.1 Digitaalinen ääni

Jotta voidaan ymmärtää äänimoottorin oleellisimpia toimintoja, on tiedettävä mitä digitaalinen ääni oikeastaan on. Digitaalisen äänen rakenteen ymmärtäminen ja käsittely ovat oleellisia osia äänimoottorin suunnittelua.

Digitaalinen ääni koostuu useista näytteistä. Nämä näytteet on kvantisoitu eli pyöristetty ja muunnettu binäärimuotoisiksi. Näytteiden määrä suhteessa äänenkestoon vaikuttaa äänenlaatuun todella paljon: mitä enemmän näytteitä on, sitä parempi on äänenlaatu. [2]

Pulssikoodimodulaatio eli PCM on yleisin digitaalisen äänen muoto. Pulssikoodimodulaatiota käytetään esimerkiksi digitaalisessa lankapuhelinverkossa. PCM-äänestä on myös olemassa monia muunnelmia. Tunnetuimmat muunnelmat ovat differentiaalinen PCM (DPCM) sekä adaptiivinen PCM (ADPCM).

PCM-ääni rakentuu analogisesta äänestä otetuista näytteistä. Näytteet otetaan tasaisin väliajoin. Näytteiden taso voidaan ilmaista lineaarisesti tai logaritmisesti. [3]

2.2 Erilaisia äänimoottoreita ja niiden ominaisuuksia

Erilaisia äänimoottoreita on paljon, ja onkin projektin onnistumisen kannalta todella tärkeää selvittää, millaisia äänimoottoreita on olemassa ja miten ne ja niiden toteutustavat eroavat toisistaan. Tästä voi oppia todella paljon hyödyllistä informaatiota oman äänimoottorin suunnitteluun.

Otan tähän projektiin tutkittavaksi muutaman käytetyimmän ja tunnetuimman äänimoottorin. Suositut äänimoottorit ovat todella hyvä tutkimuskohde, koska niistä voi saada selville ne ominaisuudet, joiden takia ne ovat suosittuja. Näitä ominaisuuksia on tarkoitus sitten mahdollisuuksien mukaan toteuttaa omaan äänimoottoriin.

Äänimoottorin tutkiminen toteuttamalla testiohjelma tai valmiin ohjelman lähdekoodin lukeminen auttaa myös todella paljon hahmottamaan, miten kukin äänimoottori toimii ja miten niitä käytetään. Testiohjelmalla tässä tarkoitetaan sellaista ohjelmaa, joka esimerkiksi toistaa musiikkia ja jotain toista ääntä samaan aikaan käyttäen jotain tiettyä äänimoottoria.

2.2.1 Audiere

Audiere on korkean tason ääniohjelmistorajapinta, joka pystyy toistamaan useita erityyppisiä ääniformaatteja. Äänentoistamiseen Audiere käyttää Windowsin puolella DirectX:n DirectSound–kirjastoa tai vaihtoehtoisesti WinMM-kirjastoa. Linuxin ja Macin puolella äänentoisto hoidetaan joko OSS:n (Open Sound System) tai Cygwinin kautta. [4]

Audiere on todella mielenkiintoinen äänimoottori, koska se on avointa lähdekoodia ja osaa toistaa useita eri formaatteja. Audieren helppokäyttöisyys on

myös suuri etu. Testiohjelman kirjoittaminen Audierea käyttäen onnistui muutamassa minuutissa.

Huonoa Audieressä on se, että siitä puuttuu aivan kokonaan tuki 3D-äänien toistamiselle. Tämä voi olla suuri haitta pelien kohdalla, koska niissä monesti halutaan liikkua 3D-maailmassa. Tämän lisäksi Audieren kehitys on pysähtynyt kokonaan tai sitten se on todella hidasta, koska viimeinen versio on tullut keväällä 2006 [4].

2.2.2 OpenAL

OpenAl on Creativen ja Applen ylläpitämä ääniohjelmointiin tarkoitettu ohjelmointirajapinta. Se on suunniteltu pelien käyttöön, ja se onkin käytössä monissa kaupallisissa peleissä. Vaikka OpenAl on pääasiallisesti pelikäyttöön suunniteltu, sitä voi myös käyttää ohjelmiin, joissa vaaditaan täydellisempää äänenhallintaa tai 3D-ääntä. OpenAl on avointa lähdekoodia, ja se on lisensoitu LGPL-lisenssin alle. Alun perin OpenAL oli Loki software -nimisen yrityksen aloittaman projekti, mutta se siirtyi myöhemmin Creativen omistukseen. [5]

OpenAL pitää sisällään kaikki toiminnot, mitä pelien ääniohjelmointiin voi tarvita. Siltä onnistuu 3D-äänentoisto, Doppler-efektin mallintaminen ja monien muiden efektien lisääminen ääneen. [5]

Huono puoli OpenAL:ssä on sen monipuolisuuden tuoma monimutkaisuus. Sen käytön opetteleminen ei onnistu muutamassa minuutissa, kuten esimerkiksi Audieren. Dokumentaatiota on kuitenkin melko paljon, mutta se vaikuttaa olevan suureksi osaksi vanhentunutta. Mukana tulevassa dokumentaatiossa olevat esimerkit eivät toimineet enää. Mukana tulee kuitenkin malliohjelmia, joista saa hyviä esimerkkejä, miten OpenAL:ää voi käyttää.

2.2.3 BASS

BASS-äänimoottori on usealla alustalla toimiva kaupallinen äänimoottori, jota voi laajentaa erilaisten ohjelmistolisäkkeiden avulla. BASS-äänimoottoria voi käyttää ilmaiseksi vapaasti levitettäviin ohjelmiin, mutta kaupallisia ohjelmia varten on ostettava lisenssi. Äänimoottorin ominaisuuksiin kuuluvat esimerkiksi useiden ääniformaattien tuki, verkon yli äänen virtauttaminen, useiden äänikorttien tai äänipiirien samanaikainen käyttäminen ja erilaiset efektit. [6]

BASS-äänimoottorin mukana tulee hyvä määrä dokumentaatiota, jonka lisäksi mukana tulee useita esimerkkiohjelmia.

2.2.4 FMOD Ex

FMOD Ex on kaupallinen äänimoottori, joka on todella monipuolinen. Se mm. tukee yli 20 erilaista ääniformaattia. Äänimoottoria saa käyttää ilmaisohjelmiin vapaasti, mutta muita ohjelmia varten tarvitaan lisenssi. FMOD Ex -äänimoottoria voidaan käyttää C-, C++-,C#-, Delphi- ja Visual Basic -ohjelmointikielten kanssa, jonka lisäksi se myös toimii useassa eri ympäristössä. [7]

FMOD Ex –äänimoottorin käyttämiselle on melko vähän esimerkkejä, mutta sen mukana tulee melko hyvä dokumentaatio.

2.2.5 XACT

XACT on Microsoftin alun perin Xboxille suunnittelema äänimoottori. Tämä äänimoottori tuli myös PC ympäristöön käyttöön DirectX:n 2008 SDK:n mukana. XACT:n ominaisuuksiin kuuluvat tuki WAV- ,AIFF- ja WMA-tiedostoille, tuki 5.1-kaiutinjärjestelmälle, useiden äänien ryhmittely erilaisiksi paketeiksi, jotka sisältävät mm. asetuksia äänille. XACT:hen kuuluu myös graafinen työkalu, jonka avulla ääniä voi muokata ja yhdistellä. [8]

Oman testiohjelman kirjoittaminen XACT:n kanssa onnistuu melko helposti koska tarjolla on paljon hyviä esimerkkejä. XACT sisältää kuitenkin hyvin suuren määrän erilaisia toimintoja, jonka takia sen sujuvan käytön opettelu vie melko paljon aikaa. Lisäksi XACT:n mukana tulevan graafisen työkalun opetteluun menee jonkin verran aikaa.

2.3 Äänimoottoreiden vertailu

Äänimoottoreita on kehitetty monenlaisia, kuten edellisesti luvusta käy ilmi. Jotkut niistä sopeutuvat paremmin massiivisten kaupallisien pelien kehittämiseen ja jotkut sitten taas pieniin vapaasti levitettäviin peleihin.

Kaikille tässä opinnäytetyössä tutkituille äänimoottoreille on yhteistä se, että ne ovat dynaamisesti linkitettäviä kirjastoja. Dynaamisesti linkitettävät kirjastot ovat hyvä tapa saada ohjelmiin toimintoja kirjoittamatta niitä suoraan ohjelman sisälle. Tällä tavalla vältetään isoilta ohjelmistomuutoksilta, vaikka jotain osia kirjastossa päivitettäisi tai korjattaisi.

Taulukossa 1 on yksinkertainen vertailu tutkittujen äänimoottoreiden ominaisuuksista. Kuten tästä taulukosta näkee, on 3D-äänten tuki hyvin tavallinen äänimoottoreiden ominaisuus. Audiere on ainoa äänimoottori, tutkittavista äänimoottoreista, josta se puuttuu. Taulukossa on otettu huomioon myös OGG Vorbis –tiedostojen tuki, koska se toteutetaan tämän opinnäytetyön äänimoottorin. Tätä tiedostomuotoa tukevat kaikki muut äänimootorit, paitsi Microsoftin XACT-äänimoottori, joka tukee ainoastaan Microsoftin omia ääniformaatteja. Äänimoottorin laajennettavuus on myös ominaisuus, mikä tuo todella paljon lisäarvoa äänimoottorille, mutta voi olla hankala toteuttaa. Tämä ominaisuus onkin vain yhdellä tarkastelluista äänimoottoreista. Taulukossa käy myös ilmi, miten helppo kutakin äänimoottoria on käyttää. Helppokäyttöisyys ilmoitetaan asteikolla 1 – 5, jossa 1 tarkoittaa vaikeata ja 5 helppoa.

Taulukko 1. Äänimoottoreiden vertailu

	Audiere	OpenAL	BASS	FMOD Ex	XACT
3D-äänet		*X*	X	X	X
Efektejä	X	*X*	X	X	X
Tuki OGG Vorbis -tiedostoille	X	*X*	X	X	
Laajennettavissa ohjelmalisäkkeillä			X		
Helppokäyttöisyys asteikko (1 – 5)	5	*1*	2	4	3

Erilaisten äänimoottoreiden tutkiminen on tärkeää ennen oman äänimoottorin suunnittelua, koska muista toteutuksista voi oppia todella paljo. Äänimoottoreiden kokeileminen myös käytännössä antaa hyviä ideoita siihen, miten oman äänimoottorin käyttö on hyvä toteuttaa. Muista äänimoottoreista voi myös olla hyötyä ongelmien toteutuksessa vastaan tulevien ongelmien ratkaisemisessa, koska niistä näkee miten muut ovat kyseisen ongelman ratkaisseet.

3 ÄÄNIMOOTTORIN SUUNNITTELU

Tarkka suunnittelu on todella tärkeä osa mitä tahansa projekti. Se varmistaa projektin selkeän etenemisen päätöspisteeseen asti. Tässä opinnäytetyössä suunnittelua helpottaa se, että ominaisuuksia ja toteutustapoja voi verrata muihin olemassa oleviin äänimoottoreihin.

3.1 Vaatimusmäärittely

Äänimoottorin tärkeimmät ominaisuudet tässä projektissa ovat seuraavat:

- useiden äänien toistaminen samanaikaisesti
- taustamusiikin toistaminen muiden äänien lisäksi
- ohjelmointirajapinnan helppokäyttöisyys
- OGG Vorbis -muotoisten äänitiedostojen toistokyky
- 3D-äänien toistokyky.

Näiden ominaisuuksien lisäksi tarkoitus on saada ainakin joitain seuraavista ominaisuuksista myös toimiviksi:

- jokaisen äänen erillinen äänenvoimakkuuden säätäminen
- mahdollisuus jakaa useita ääniä omiksi ryhmikseen, jolloin voidaan esimerkiksi taustamusiikkeihin kuuluvien äänien voimakkuus säätää vaikuttamatta ollenkaan muihin ääniin.
- äänenkorkeuden muuttaminen toistettavilla äänillä toistamisen aikana ja ennen sitä
- yksinkertainen tilakaiku, jonka voi asettaa kaikille äänille, ääniryhmälle tai yksittäiselle äänelle.

Kaikkien edellä mainittujen kohtien täyttyminen ei ole välttämätöntä, että kasassa olisi täysin toimiva ja käyttökelpoinen äänimoottori, koska yksinkertainenkin äänentoistokyky helppokäyttöisen ohjelmointirajapinnan kautta tekee jo äänten käyttämisen mahdolliseksi useissa erilaisissa ohjelmissa.

3.2 Äänimoottorin yhteys pääohjelmaan

Äänimoottori toteutetaan jaettuna kirjastona (DLL-tiedosto), eli pääohjelman täytyy pystyä käyttämään jaettuja kirjastoja, että keskustelu äänimoottorin kanssa olisi mahdollista.

Jaetuilla kirjastoilla suurena etuna on, se että komponentteja voidaan yksitellen päivittää ilman, että koko ohjelmaa tarvitsisi asentaa uudestaan. Tämä tekee myös erilaisten muutosten testaamisesta paljon helpompaa, koska ohjelman hakemistosta tarvitsee vain yksi tiedosto vaihtaa. Tämän mahdollistaa se, että jaetut kirjastot voidaan ladata ohjelman käytettäväksi dynaamisesti joka kerta uudestaan kun ohjelma käynnistyy [9].

3.3 Ääniformaattien tuki

Ensimmäisiin kokeiluihin äänimoottorin kehityksessä käytetään satunnaisesti generoitua dataa, jotta nähdään, saadaanko data oikealla tavalla kulkemaan äänikortille asti. Tämän jälkeen kirjoitetaan luokka, jonka avulla voidaan lukea WAV-tiedostoja. WAV-tiedostot soveltuvat hyvin ensimmäiseksi testattavaksi ääniformaatiksi, koska niistä löytyy todella paljon tietoa sekä niiden lukeminen on suhteellisen yksinkertaista.

Kun edellä mainitut asiat on saatu toteutettua onnistuneesti, lisätään äänimoottoriin tuki OGG Vorbis –tiedostoja varten. Tämä on todella tärkeä tuki, koska tiedostojen koko on huomattavasti pienempi kuin pakkaamattomien WAV-tiedostojen.

4 TYÖKALUT JA TEKNOLOGIAT

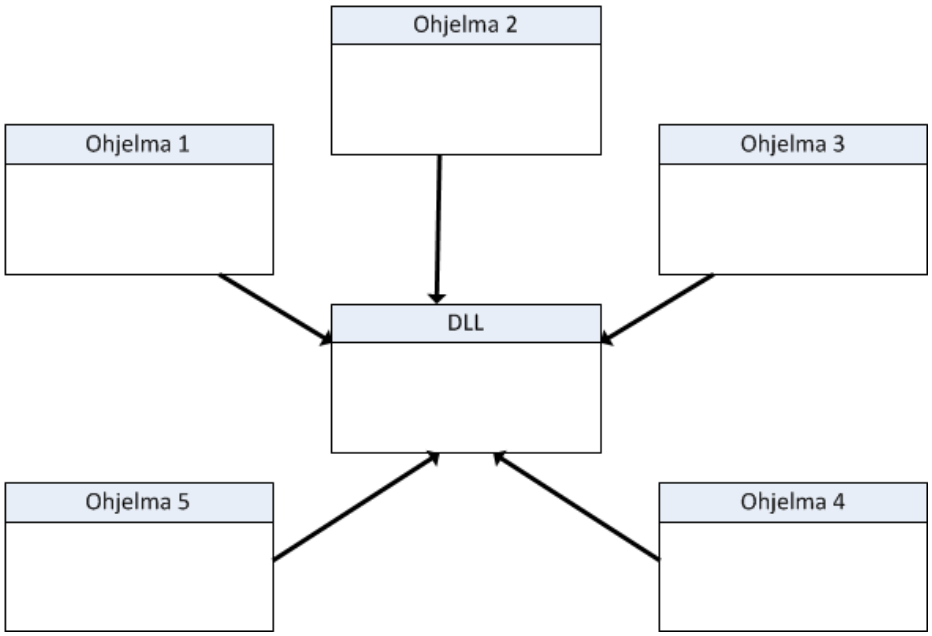
Äänimoottori toteutetaan tässä opinnäytetyössä C++-ohjelmointikielellä, jonka ohjelmointiin käytetään Microsoftin Visual Studio 2010 -ohjelmointiympäristöä. Visual Studio tarjoaa monipuolisen ohjelmointiprojektin hallinnan ja erittäin hyvän ohjelmointiympäristön, jossa on mukana mm. virheenjäljitystyökalut.

4.1 Dynaamiset linkkikirjastot

Dynaamiset linkkikirjastot (engl. Dynamic-link library) ovat Microsoftin toteuttama tapa toteuttaa jaetun kirjaston konsepti. Nämä kirjastot ovat Windowsissa useimmiten DLL-päätteisiä tiedostoja, joten niistä puhutaan useimmiten vain DLL-tiedostoina. Windows-ympäristössä on myös muunlaisia jaettuja kirjastoja, näitä ovat esimerkiksi OCX-tiedostot, jotka ovat ActiveX-komponentteja sisältäviä kirjastoja, sekä DRV-tiedostot, joita on käytetty ajuritiedostoina. [9]

DLL-tiedostot ovat käytännössä samanlaisia binääritiedostoja kuten normaalit ohjelmatiedostot eli EXE-tiedostot. Molemmat tiedostot ovat rakenteeltaan PE-tyyppisiä (Portable Executable) tiedostoja. Vaikka DLL-tiedostot ovat rakenteellisesti hyvin samanlaisia kuin EXE-tiedostot, ei niitä kuitenkaan voi suorittaa yksistään, vaan tarvitaan joku ohjelma, joka käyttää kyseistä DLL-tiedostoa. [9]

Suurin hyöty, mikä saavutetaan DLL-tiedostoista, on koodin uudelleen käytettävyys. Monissa ohjelmissa voidaan käyttää samaa DLL-tiedostoa, jolloin sen sisältämää toiminnallisuutta ei tarvitse uudelleenkirjoittaa ohjelmaan ollenkaan. DLL-tiedostot voivat ohjelmarutiinien lisäksi sisältää monenlaista dataa. Niissä voi esimerkiksi olla dataa, kuvia tai vaikka ääntäkin. Kuvassa 2 on esimerkki yhdestä DLL-tiedostosta, jota useat eri ohjelmat käyttävät. [9]



Kuva 2. Useat ohjelmat käyttävät samaa DLL-tiedostoa.

4.2 XAudio2

Äänimoottorin ydin tässä projektissa tulee olemaan XAudio2. Se on valittu siksi, että XAudio2 on DirectX:n tämän hetkinen ääniohjelmointiin tarkoitettu rajapinta. Se on DirectSound-rajapinnan seuraaja. Käytännössä suurin osa Windows-ympäristössä toimivista peleistä käyttää joko DirectX:n, nyt jo vanhentunutta, DirectSound-rajapintaa tai sitten DirectX:n XAudio2-rajapintaa.

XAudio2 on matalan tason ääni-ohjelmointirajapinta, joka on tarkoitettu erityisesti suunniteltu tehokkaiden ja monipuolisten peliäänimoottoreiden toteuttamiseen. Sen ominaisuuksiin kuuluvat mm. erilaiset efektit, äänien yhdistäminen alasmiksausäänillä, pakatun ADPCM-äänen tuki ja surround-äänen tuki. [10]

4.3 X3DAudio

X3DAudio on apukirjasto, jota hyödynnetään XAudio2:n kanssa silloin, kun halutaan sijoittaa ääniä 3D-ympäristössä ja saada äänet kuulostamaan siltä, että ne tulisivat jostain tietystä suunnasta. Illuusio äänen tulosuunnasta luodaan

käyttämällä sellaisia rakenteita kuten äänenlähettäjä (engl. emitter) ja kuuntelija (engl. listener). Näille rakenteille määritellään koordinaatit 3D-maailmassa, jonka jälkeen X3DAudio voi laskea äänelle tarvittavat muutokset. [11]

4.4 WAV-tiedostot

WAV-tiedostomuoto on Microsoftin ja IBM:n kehittämä tiedostomuoto digitaalisen äänen tallentamiseen. WAV on nykyään kaikista tuetuin äänen tallentamiseen käytetty tiedostomuoto. Tähän vaikuttaa todennäköisesti se, että Windows tukee sitä natiivisti sekä se, että WAV-tiedostojen lukeminen on melko yksinkertaista. [12]

RIFF-tiedostomuoto

WAV-tiedostot ovat rakenteellisesti RIFF-tiedostoja, eli Resource Interchange File Format –tiedostoja (Resurssin vaihto tiedostomuoto). Ne koostuvat palasista (engl. chunks). Jokaisella palasella on oma tarkoitus tiedoston tulkitsemisessa. Palasen alussa on aina ensimmäisenä tunniste, joka koostuu neljästä merkistä. Tätä tunnistetta sanotaan Windows-käyttöjärjestelmän puolella FourCC-tunnisteeksi (Four Character Code). Taulukossa 2 on pakkaamattoman WAV-tiedoston rakenne, josta on helppo hahmottaa RIFF-tiedostomuodon eri palaset. Taulukon avulla voi myös kirjoittaa ohjelman, joka lukee PCM-äänen WAV-tiedostosta.

Taulukko 2. WAV-tiedoston rakenne.

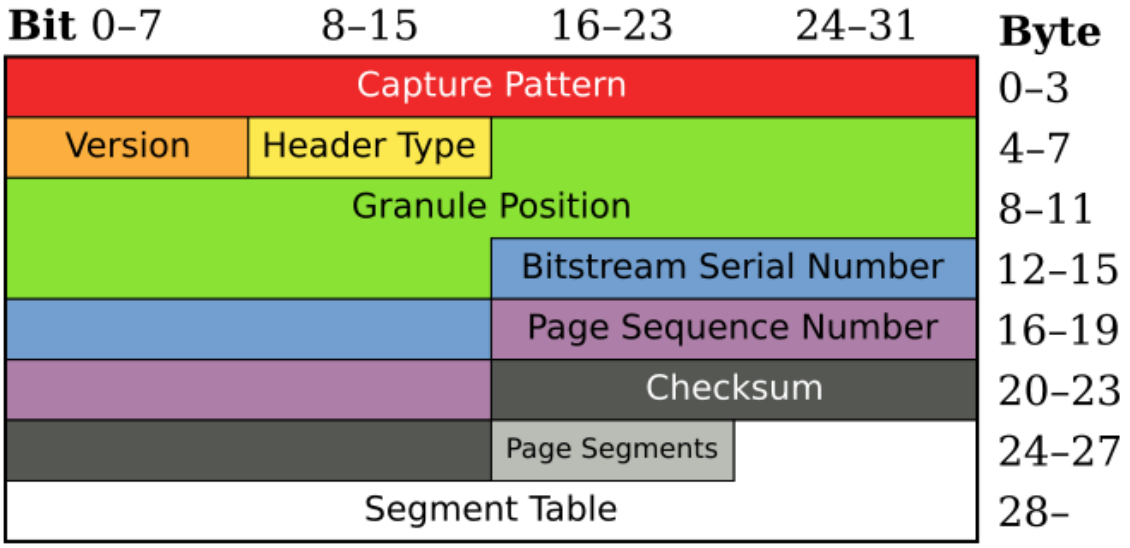
WAV-tiedoston rakenne			
Etäisyys tiedoston alusta (tavuina)	**Kentän nimi**	**Kentän koko (tavuina)**	**Kuvaus**
0	ChunkID	4	RIFF-tiedoston ensimmäinen palanen, joka kertoo minkälaista dataa tiedostossa on. Tunniste on "WAVE".
4	ChunkSize	4	
8	Format	4	
12	SubChunk1 ID	4	WAV-tiedoston "fmt "-palanen (välilyönti kuuluu palasen tunnisteeseen). Tämä palanen määrittelee tiedostoon tallennetun äänen ominaisuudet.
16	SubChunk1 Size	4	
20	AudioFormat	2	
22	NumChannels	2	
24	SampleRate	4	
28	ByteRate	4	
32	BlockAlign	2	
34	BitsPerSample	2	
36	SubChunk2 ID	4	RIFF-tiedoston "data"-palanen. Digitaalinen ääni on tallennettu tähän palaseen.
40	SubChunk2 Size	4	
44	Data	x	

WAV-tiedostoissa voi myös tallentaa pakattua ääntä. Pakattua ääntä sisältävissä WAV-tiedostoissa on erillinen palanen, josta selviää miten pakattu äänidata voidaan purkaa. Tämän palasen tunniste on normaalisti ”fact”. Ilman ”fact”-palasta pakattua äänidataa ei voida purkaa, jolloin sitä ei voida toistaa oikein. [13]

4.5 Ogg Vorbis -tiedostot

Ogg on Xiph.org yhdistyksen ylläpitämä avoimen standardin multimediasäiliötiedostomuoto. Ogg-tiedostoihin voidaan säilöä lähes mitä vain multimediaa. Koska Ogg on vain säiliötiedostomuoto, on sen sisällä oleva data usein pakattu jollain koodekilla joskus jopa useilla koodekeilla. Tästä esimerkkinä on Ogg-tiedostot, jotka sisältävät videota, ääntä sekä tekstityksen videoon. Tällaisessa tiedostossa voi nämä kaikki ominaisuudet olla pakattu eri koodekilla. [14]

Ogg-tiedostot koostuvat palasista, joita kutsutaan Ogg-sivuiksi. Jokainen tällainen Ogg-sivu alkaa tunnisteella ”OggS”. Kuvassa 3 on Ogg-sivun rakenne. Ogg-sivu alkaa ”Capture Pattern”-kentällä, joka on Ogg-sivun tunniste. Seuraavana tulee versionumero, joka on nykyään turha, koska Ogg-sivuja on ainoastaan yhdenlaisia. ”Header Type”-kenttä ilmoittaa ollaanko mediavirtauksen keskellä, alussa vai lopussa. ”Granule Position”-kenttä sisältää esitysaikaleiman, jota käytetään datavirran synkronointiin. Seuraava kenttä on sarjanumero, jonka avulla tunnistetaan mihin datavirtaan kyseessä oleva sivu kuuluu. ”Page Sequence Number”-kenttä on sivunumero samaan tapaan kuin normaalissa kirjassakin, mutta sitä käytetään enimmäkseen tarkastamaan puuttuuko Ogg-tiedostosta sivuja. ”Checksum”-kenttä on CRC-tarkistussummaa varten. Seuraava kenttä ilmoittaa, montako lohkoa Ogg-sivussa on. Lopuksi tulee lohkotaulukko, joka voi pitää sisällään enintään 255 lohkoa. [15]



Kuva 3. Ogg-sivun rakenne [15].

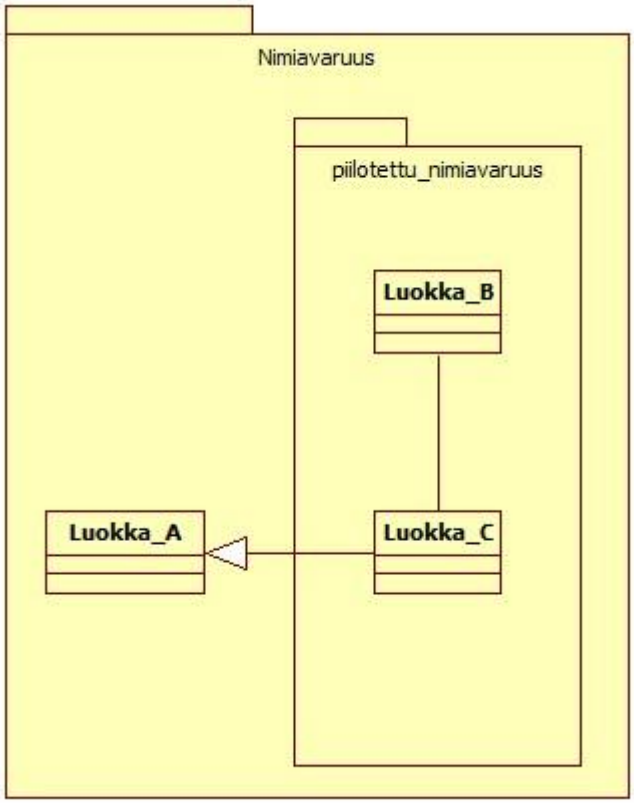
Tässä opinnäytetyössä kuitenkin perehdytään vain Ogg-tiedostoihin pakattuun ääneen. Useimmiten Ogg-tiedostojen äänen pakkaukseen käytetään Vorbis-koodekkia, joka on myös avointa lähdekoodia. Ogg Vorbis –tiedostoja käytetään mm. kaupallisissa peleissäkin. Esimerkiksi Grand Theft Auto: San Andreas ja Guitar Hero: On Tour käyttävät Ogg Vorbista peliäänten formaattina. [16]

5 ÄÄNIMOOTTORIN TOTEUTUS

5.1 Ohjelmointirajapinnan toteuttaminen

Ohjelmointirajapinnan tarkka suunnittelu on tärkeää, koska yksi vaatimusmäärittelyssä huomioiduista välttämättömistä ominaisuuksista oli helppokäyttöinen ohjelmointirajapinta. Ohjelmointirajapinnan suunnittelu on iso osa moottorin suunnittelua. Tämä osa moottoria määrittelee sen, miten äänimoottoria tullaan käyttämään muissa ohjelmissa.

Koska äänimoottorin rakenne koostuu useista luokista, on hyvä rajata äänimoottorin osat muusta ohjelmasta. Tämä toteutetaan luomalla äänimoottorin ympärille nimiavaruus. Nimiavaruus rajaa äänimoottorin kaikki komponentit, siten että se on helpompi hahmottaa kokonaisuutena. Tämän nimiavaruuden lisäksi luodaan äänimoottorin sisälle vielä toinen nimiavaruus, joka rajaa käyttäjältä pois pelkästään äänimoottorin sisäiseen käyttöön suunnitellut komponentit. Äänimoottorin käyttäjälle jätetään näkyville ainoastaan ne luokat, joita on pakko suoraan käsitellä. Kuvan 4 UML-kaaviossa on vastaavanlainen tilanne. Kuvan luokka A on käytössä nimiavaruudessa, mutta luokka B ja C ovat ainoastaan näkyvissä piilotetussa nimiavaruudessa, joka on ensimmäisen nimiavaruuden sisällä.



Kuva 4. Nimiavaruudet UML-kaaviossa

Luokat, jotka näkyvät äänimoottorin ensimmäisessä nimiavaruudessa, on syytä pitää mahdollisimman yksinkertaisina ja selkeinä. Tämä auttaa ohjelmointirajapinnan käyttäjää oppimaan nopeasti käyttämään ohjelmointirajapintaa oikein. Olio-ohjelmoinnin keinot auttavat siirtämään osan toiminnoista luokkiin, jotka eivät normaalisti käyttäjälle näy. Kaikki toiminnallisuudet ja ominaisuudet, joita käytetään useammassa kuin yhdessä paikassa, on hyvä siirtää tällaisiin abstrakteihin luokkiin. Abstrakteja luokkia ei voida instantioida, vaan ne toteutetaan aina jossain toisessa luokassa.

5.2 Äänigraafin toteuttaminen

Äänigraafi on äänimoottorin toiminnan kuvaus. Äänigraafi määrittelee sen, miten ääni syntyy ja mitä kautta se kulkee äänipiirille ja kaiuttimista ulos. Äänigraafi toteutetaan XAudio2:n äänikomponenttien avulla. Näihin kuuluu seuraavat komponentit:

- äänilaite
- masterointiääni (engl. Mastering voice)
- alasmiksausääni (engl. Submix voice)
- lähdeääni (engl. Source voice)
- äänidata.

Näistä kaikki muut komponentit ovat pakollisia paitsi alasmiksausääni. Alasmiksausääni onkin helpottamassa äänen käsittelyä. Tätä ominaisuutta ei ollut XAudio2:n edeltäjässä DirectSoundissa. XAudio2:n äänigraafin komponentit on toteutettu niin, että äänen prosessointi tapahtuu aina erillisessä säikeessä. XAudio2:n äänigraafiin voi myös vaikuttaa dynaamisesti ajon aikana, eli mm. lähdeäänten kohteita voidaan vaihtaa tai äänille voidaan lisätä efektejä. [17]

Äänilaite

Äänilaite tarkoittaa tietokoneessa olevaa äänikorttia tai integroitua äänipiiriä. Äänilaitteelle ääni tulee masterointiäänen kautta. Äänilaitteesta ääni siirtyy johtoja pitkin vahvistimelle ja sitä kautta kaiuttimille.

Masterointiääni

Masterointiääni edustaa äänilaitetta, jonka kautta ääntä toistetaan. Tälle komponentille ei voida suoraan lähettää äänidataa, vaan sen pitää tulla joko alasmiksausäänen kautta tai lähdeäänen kautta. [18]

Koodiesimerkissä 1 on esitetty yksinkertainen tapa alustaan XAudio2 käytettäväksi sekä luoda masterointiääni. Koodiesimerkissä ensin alustetaan XAudio2 käyttöä varten, mikä tarkoittaa käytännössä myös, tässä esimerkissä, oletusäänilaitteen käyttöönottamista. Kun XAudio2:n alustus on suoritettu, voidaan

masterointiääni luoda. Masterointiääni luodaan XAudio2:n CreateMasteringVoice-funktiolla, jolle annetaan parametriksi viittaus masterointiääneen.

Koodiesimerkki 1. XAudio2:n alustaminen ja masterointiäänen luonti.

```
HRESULT hr;
// Initialisoidaan XAudio2
CoInitializeEx( NULL, COINIT_MULTITHREADED );
IXAudio2* pXAudio2 = NULL;

if (FAILED( hr = XAudio2Create( &pXAudio2 )))
{
    cout << “Failed to init XAudio2 engine: “ << hr << endl;
    CoUninitialize();
    return 0;
}
// Luodaan masterointiääni
IXAudio2MasteringVoice* pMasteringVoice = NULL;

if (FAILED( hr = pXAudio2->CreateMasteringVoice( &pMasteringVoice )))
{
    cout << “Failed creating mastering voice:  “ << hr << endl;
    if (pXAudio2)delete pXAudio2;
    CoUninitialize();
    return 0;
}
```

Alasmiksausääni

Alasmiksausääni on äänigraafin komponentti, jolla voidaan useita lähdeääniä yhdistää yhdeksi alasmiksausääneksi. Tämä alasmiksaus ääni voidaan lähettää joko suoraan masterointiäänelle tai sitten toiselle alasmiksausäänelle. Alasmiksausääntä voidaan käyttää myös erilaisten efektien toistamiseen, voidaan esimerkiksi toistaa alkuperäistä ääntä ja alasmiksausäänen kautta efektoitua ääntä samaan aikaan. [18]

Koodiesimerkissä 2 on yksinkertainen tapa luoda alasmiksausääni. Tämän esimerkin alasmiksausääni voi ottaa vastaan useita ääniä, jotka se lähettää masterointiäänelle. Alasmiksausäänen luovalle funktiolla annetaan tässä esimerkissä parametrina viittaus alasmiksausääneen, alasmiksauksen käyttämä kanavien määrä sekä äänentaajuus.

Koodiesimerkki 2. Alasmiksausäänen luonti.

```
IXAudio2SubmixVoice * pSFXSubmixVoice;
pXAudio2->CreateSubmixVoice(&pSFXSubmixVoice,1,44100);
```

Lähdeääni

Lähdeääni on äänigraafin aloituskohta. Tälle komponentille syötetään äänidata ja äänen ominaisuudet. Se myös voi muuttaa ääntä jonkin verran, esimerkiksi muuttamalla äänenkorkeutta, muuttaa jokaisen kanavan äänenvoimakkuutta yksitellen ja lisäämällä siihen erilaisia efektejä. Lähdeääni tarvitsee kuitenkin vähintään masterointiäänen, jolle se voi lähettää äänidatan, muuten ääntä ei lähetetä äänilaitteelle asti. [18]

Koodiesimerkissä 3 on yksinkertainen tapa luoda lähdeääni, joka on suoraan liitetty masterointiääneen. Ennen kuin voidaan lähdeääni luoda, on tiedettävä minkälaista ääntä sillä tullaan toistamaan. Tieto äänen ominaisuuksista ja viittaus lähdeääneen annetaan parametrina funktiolle, joka luo lähdeäänen.

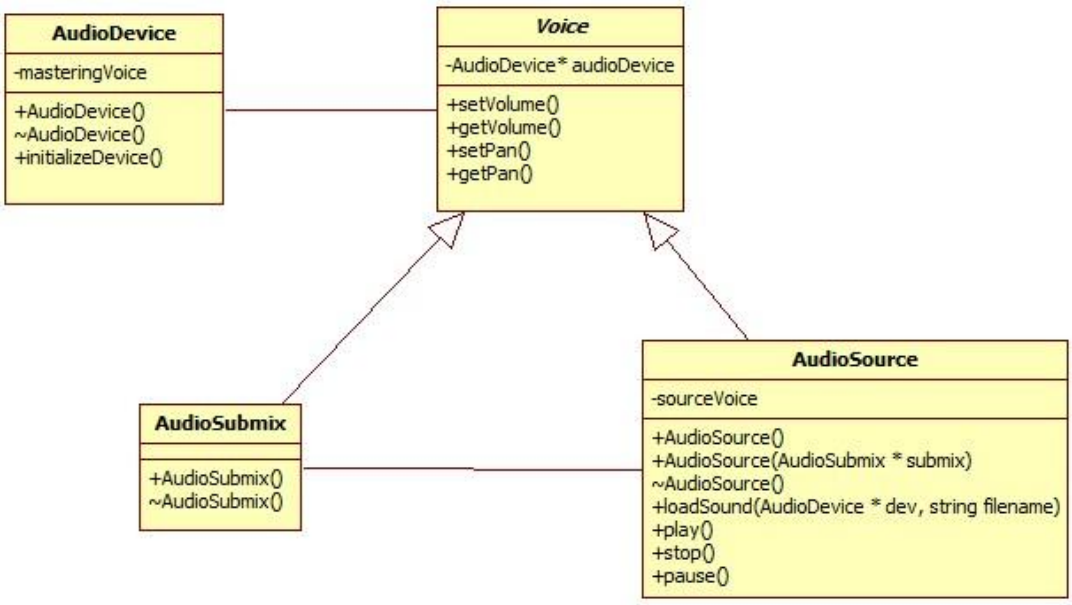
Koodiesimerkki 3. Lähdeäänen luonti.

```
// luodaan lähdeääni
IXAudio2SourceVoice* pSourceVoice;
if( FAILED( hr = pXaudio2->CreateSourceVoice( &pSourceVoice, pwfx ) ) )
{
    cout << “Error “ << hr << “ creating source voice” << endl;
    return hr;
}
```

Äänigraafin komponenttien yhdistäminen

Äänigraafin komponentit tämän opinnäytetyön äänimoottorissa yhdistetään siten, että lähes jokaista komponenttia varten tehdään oma luokka. Tämä on järkevää, koska jokaisella äänigraafin komponentilla on omanlainen alustustapansan sekä erilaisia arvoja, jotka pitää tallentaa komponentin käytön ajaksi. Kuvassa 5 on äänimoottorin äänigraafista yksinkertainen luokkakaavio. Tästä kaaviosta on helppo hahmottaa, miten äänigraafi voidaan käytännössä toteuttaa ja miten eri komponentit linkittyvät toisiinsa. Kuvan AudioDevice-luokka pitää sisäl-

lään masterointiäänen ja kaikki XAudio2:n alustukseen sisältyvät toimenpiteet. Voice-luokka on abstrakti luokka, jossa on alasmiksausäänelle ja lähdeäänelle yhteisiä toimintoja ja ominaisuuksia. AudioSource- ja AudioSubmix-luokka toteuttavat Voice-luokan. AudioSubmix-luokka on hyvin pieni luokka, joka on tarkoitettu ainoastaan äänien ryhmittelyä varten. Esimerkiksi voidaan jakaa musiikkeihin ja ääniefekteihin kuuluvat äänet omiin ryhmiinsä. AudioSource-luokka edustaa ääntä, joka halutaan toistaa. Se sisältää lähdeäänen ominaisuudet ja alustustoimenpiteet. Tämän luokan kautta ladataan ja toistetaan kaikki äänet. Kuvan luokkakaavio ei sisällä kaikkia tarvittavia atribuutteja tai metodeja, vaan sen tarkoitus on vain havainnollistaa audiograafin komponenttien yhteyksiä äänimoottorissa.



Kuva 5. Audiograafin toteuttaminen äänimoottorissa.

5.3 PCM-audion toistaminen

PCM-audion eli pakkaamattoman digitaalisen äänen toistaminen toteutetaan lähdeäänellä. Lähdeäänelle annetaan luontivaiheessa parametriksi ääni-informaatiotietue, jonka avulla saadaan luotua oikeanlainen lähdeääni. Tämän lisäksi tarvitaan äänidata. Äänidata syötetään lähdeäänelle XAudio2:n omalla

äänipuskuritietueella, joka on XAUDIO2_BUFFER-tietue. Kun molemmat tietueet ovat lähdeäänellä ja äänigraafi on muilta osin kunnossa, voidaan ääntä alkaa toistamaan kutsumalla lähdeääni-olion start-metodia.

5.3.1 Ääni-informaatiotietue

Ääni-informaatiotietue eli WAVEFORMAT-tietue pitää sisällään kaiken tarpeellisen tiedon äänentoistamista varten. Tätä tietuetta käytetään äänentoistamiseen XAudio2:lla ja sitä edeltäneellä DirectSoundilla. Taulukossa 3 on esitetty tähän rakenteeseen kuuluvat osat niiden tyypit sekä kuvaus niistä.

Taulukko 3. Ääni-informaatiotietue.

Muuttujan tyyppi	**Muuttujan nimi**	**Kuvaus**
WORD	wFormatTag	Määrittelee äänen tyypin. Tässä tapauksessa on aina ”WAVE_FORMAT_PCM”.
WORD	nChannels	Määrittelee kanavien määrän äänessä esimerkiksi mono-äänellä 1 ja stereo-äänellä 2.
DWORD	nSamplePerSec	Määrittelee montako näytettä sekunnin aikana äänessä on.
DWORD	nAvgBytesPerSec	Keskiarvo tiedonsiirto nopeudesta. (Tavuja sekunnisa)
WORD	nBlockAlign	Määrittelee montako tavua kutakin näytettä kohti käytetään.

Ääni-informaatiorakenteesta on myös olemassa uudempi versio, mutta sen käyttö ei ole tarpeellista tässä opinnäytetyössä toteutettavassa äänimoottorissa. Uudempi versio on nimeltään ”WAVEFORMATEX”. Uudemmassa rakenteessa on kaksi muuttujaa lisää: wBitsPerSample, joka ilmoittaa bittien lukumäärän näytettä kohti, sekä cbSize, joka määrittelee pakattuja ääniä varten olevan lisäinformaation määrän. [19][20]

5.3.2 XAudio2-puskuri

XAudio2-puskuri on myös C/C++-tietue. Tämän tietueen tärkeimmät tiedot ovat osoite äänidataan, äänidatan pituus sekä osoite ääni-informaatiotietueeseen. Näiden tietojen lisäksi tietueessa on paljon muita ominaisuuksia. Nämä ominaisuudet selviävät taulukosta 4 Taulukossa on tietueen muuttujat, niiden tyyppi ja lyhyt kuvaus muuttujan tarkoituksesta tietueessa.

Taulukko 4. XAUDIO2_BUFFER-tietue.

Muuttujan tyyppi	**Muuttujan nimi**	**Kuvaus**
UINT32	Flags	Määrittelee lisäominaisuuksia tietueelle.
UINT32	AudioBytes	Äänidatan määrä tavuina.
const BYTE*	pAudioBytes	Osoite äänidataan.
UINT32	PlayBegin	Ensimmäinen näyte, josta toistaminen aloitetaan.
UINT32	PlayLength	Määrittelee montako näytettä toistetaan.
UINT32	LoopBegin	Näyte, josta toistosilmukka aloitetaan.
UINT32	LoopLength	Toistosilmukkaan kuuluvien näytteiden määrä.
UINT32	LoopCount	Määrittelee montako kertaa määritelty silmukka toistetaan.
void*	pContext	Viittaus takaisinkutsuja sisältävään tietueeseen tai luokkaan.

Yksinkertaisen äänentoistamisen kannalta tärkeimmät kohdat ovat AudioBytes sekä pAudioBytes, koska ne määrittelevät toistettavan äänen. Muissa muuttujissa voidaan pitää nolla-arvoja, jos ei tarvita esimerkiksi takaisinkutsufunktioita tai toistosilmukkaa. [21]

Koodiesimerkissä 4 luodaan XAudio2-puskuri ja annetaan sille osoite äänidataan sekä äänidatan koko tavuina, jonka jälkeen puskuri syötetään lähdeäänelle.

Koodiesimerkki 4. XAudio2-puskurin luonti ja syöttäminen lähdeäänelle.

```
// Luodaan puskuri ja annetaan sille tieto äänidatan sijainnista sekä koosta.
XAUDIO2_BUFFER buffer = {0};
buffer.pAudioData = pbWaveData; //Äänidatan sijainti
buffer.AudioBytes = cbWaveSize; //Äänidatan koko tavuina

//Viedään puskuri lähdeäänelle
if( FAILED( hr = pSourceVoice->SubmitSourceBuffer( &buffer ) ) )
{
   cout << “Error “ << hr << “ submitting source buffer” << endl;
   pSourceVoice->DestroyVoice();
   if (pbWaveData)delete pbWaveData;
   return hr;
}

hr = pSourceVoice->Start( 0 );
```

5.4 Ohjelmalisäkearkkitehtuurin toteuttaminen

Äänimoottoriin tarvitaan jo tuki kahdelle täysin erilaiselle ääniformaatille ja tulevaisuudessa voidaan tarvita muitakin ääniformaatteja. Tämän takia ääniformaattien tuki on toteutettava siten, että äänimoottoriin on jatkossakin helppo lisätä tuki uudelle ääniformaatille ilman, että tarvitsee äänimoottoriin itsessään tehdä muutoksia. Toimivin ratkaisu tähän ongelmaan on toteuttaa äänimoottorin erilaisten ääniformaattien lukeminen ohjelmalisäkearkkitehtuurin mukaisesti.

Ohjelmalisäkearkkitehtuuri mahdollistaa ohjelman laajentamisen monin tavoin muuttamatta pääohjelman rakennetta ollenkaan. Tällä arkkitehtuurilla on useita hyviä puolia, niihin kuuluvat mm. seuraavat:

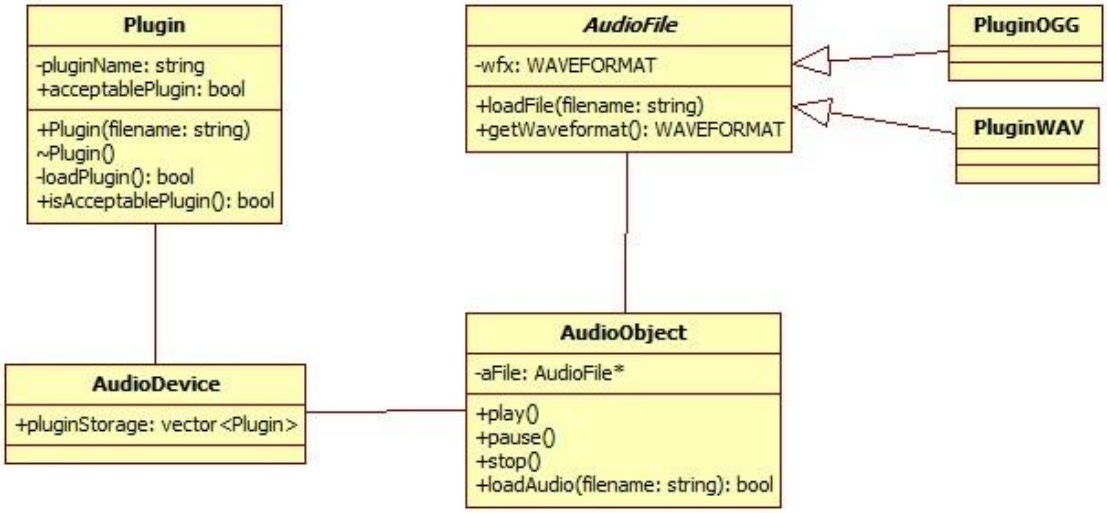
- ohjelmakoodin selkeys sekä yhtenäisyys
- ohjelman rakenne on huomattavasti modulaarisempi
- lyhyemmät käännösajat, koska kerralla tarvitsee kääntää vain yksi komponentti
- ohjelmalisäkkeen päivittämiseksi ohjelmassa, tarvitsee vain vaihtaa ohjelmalisäkkeen tiedosto, eikä pääohjelmaa tarvitse muuttaa.

Näiden lisäksi isommankin ohjelman rakenne tulee huomattavasti selkeämmäksi, ja ohjelman työstämistä on paljon helpompi jakaa useampien henkilöiden kesken tarvittaessa. [22]

Äänitiedostojen lukemista varten tarvitaan tietynlaisia toimintoja. Näihin toimintoihin kuuluu mm. tiedoston avaaminen, ääni-informaation lukeminen ja pcm-äänidatan lukeminen puskuriin. Koska kaikilla audiotiedostoilla täytyy nämä samat asiat tehdä, on tästä järkevä tehdä abstrakti luokka, jonka jokainen ääniformaattiohjelmalisäke sitten toteuttaa.

Seuraavaksi tarvitsemme luokan, joka hallitsee ohjelmalisäkkeitä. Tämä luokka pitää itsellänsä yhteyden ohjelmalisäkkeeseen eli DLL-tiedostoon ja hoitaa näiden ohjelmalisäketiedostojen lataamisen ja sulkemisen. Luokka myös tarkastaa onko avattu tiedosto yhteensopiva äänimoottoriin. Tämän luokan nimeksi on järkevä antaa ”Plugin”, koska se edustaa ohjelmalisäkkeen olemassaoloa.

Koska ohjelmalisäkkeitä voi olla samaan aikaan käytössä enemmän kuin yksi, tarvitsee niille luoda paikka, jossa niitä säilytetään. Tätä varten luodaan äänilaiteluokkaan vector-tyyppinen STL-kirjaston muuttuja, joka voi pitää sisällään Plugin-tyyppisiä olioita. Tämä muuttuja toimii ohjelmalisäkkeiden keskusvarastona. Jokainen luotu ääniolio pääsee tarkastamaan äänilaite-oliolta, onko sillä sopivaa ohjelmalisäkettä ladattavaa ääntä varten. Kuvassa 6 on yksinkertainen luokkakaavio, josta näkee tämän ohjelmistolisäkerakenteen.



Kuva 6. Ohjelmistolisäkerakenne äänimoottorissa.

Tämä rakenne toimii siten, että kun äänilaitteesta luodaan instanssi, se tutkii kotihakemistonsa DLL-tiedostot. Jos DLL-tiedostoja on ja ne osoittautuvat sopiviksi äänimoottorille, niin ne ladataan automaattisesti ohjelmalisäkevarastoon. DLL-tiedostojen tarkastus tarkoittaa niiden lataamista, jonka jälkeen tutkitaan löytyykö niistä tietyt funktiot. Jos funktiot löytyvät, ovat ne äänimoottoriin sopivia, jos ei löydy, kyseessä oleva DLL-tiedosto vapautetaan. Tämän jälkeen kun luodaan ääniobjekti ja lähdetään sille lataamaan äänitiedostoa, ääniobjekti tarkastaa, minkälaista tiedostoa yritetään ladata, tämän jälkeen etsitään ohjelmalisäkevarastosta sopivaa ohjelmalisäkettä, joka osaisi kyseessä olevan tiedoston avata ja purkaa PCM-ääneksi. Koska ohjelmalisäkkeet tässä kohtaa joutuvat tekemää tiedostojen käsittelyä mahdollisesti pitkänkin aikaa, jokaiselle ääniobjektille on luotava oma instanssi sille sopivasta ohjelmistolisäkkeestä.

5.5 Äänitiedostojen lukeminen

Äänitiedostojen lukeminen hoidetaan kullekin ääniformaatille erikseen toteutetun ohjelmalisäkkeen kautta. Tiedostojen lukemisen toteuttamisessa on oltava tarkka, koska se on prosessi, joka vaikuttaa huomattavan paljon äänimoottorin

nopeuteen ja käytettävyyteen. Tiedostojen pitää myös sulkeutua oikein eikä tiedostolle saa tapahtua mitään, mikä vahingoittaisi sitä.

Tiedostoja voidaan lukea kahdella tavalla. Tiedosto voidaan lukea joko kokonaan muistiin tai sitä voidaan lukea pala kerrallaan. Jälkimmäistä tapaa, virtauttamista, käsittellään luvussa 5.6.

5.5.1 WAV-tiedostot

WAV-tiedostojen lukeminen tarkoittaa useiden eri palasten lukemista yhdestä tiedostosta. Tämä johtuu siitä, että WAV-tiedostot ovat RIFF-tyyppisiä tiedostoja. Palaset luetaan käyttäen C++-ohjelmointikielen standardin mallikirjaston fstream-oliota. Fstream-olion avulla voidaan tiedosto lukea binaarimuodossa ja tiedoston sisällä voidaan siirtyä haluttuun paikkaan [23].

Tiedoston lukeminen aloitetaan ”WAVE”-palasesta, joka on WAV-tiedostojen ensimmäinen palanen. Tämän jälkeen luetaan ”fmt ”-palanen, jonka jälkeen viimeisenä tiedostossa on ”data”-palanen. ”data”-palanen sisältää ainoastaan palasen pituuden sekä PCM-muotoisen äänidatan.

Ääni-informaatiotietue saadaan muodostettua tiedoston ”fmt ”-palasesta. Jos tiedosto on tarkoitus virtauttaa, niin lukeminen ennen toistoa lopetetaan tähän palaseen, koska ”data”-palanen luetaan vasta ääntä toistettaessa.

5.5.2 OGG Vorbis -tiedostot

OGG Vorbis -tiedostojen lukeminen toteutetaan VorbisFile-kirjaston avulla. Tämä kirjasto on suunniteltu OGG Vorbis –tiedostojen helppoa lukemista varten. Tämä kirjasto hoitaa kaiken OGG-tiedoston lukemisen ja sen avulla voidaan silmukassa lukea koko OGG Vorbis –tiedosto PCM-muotoiseksi ääneksi. Kirjasto on hyvin dokumentoitu sekä sen käyttämiseksi löytyy paljon esimerkkejä.

Tämän opinnäytetyön äänimoottorissa suoritetaan OGG Vorbis –tiedostojen lukeminen siten, että tiedosto avataan OGG Vorbis –tiedosto FILE-muuttujalla.

Tämän jälkeen voidaan tutkia onko tiedosto oikeasti OGG Vorbis –tiedosto. Seuraavaksi tiedostosta luetaan tarvittavat informaatiot ääni-informaatiotietuetta varten, jonka jälkeen voidaan alkaa lukea varsinaista äänidataa. Tiedosto luetaan silmukan sisällä, koska OGG-tiedostot koostuvat OGG-sivuista.

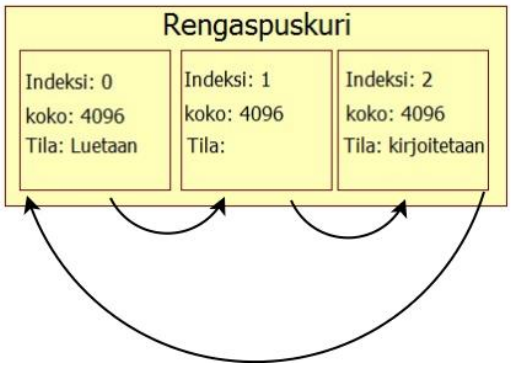
OGG Vorbis –tiedostoa virtauttaessa lukutekniikka ei juurikaan muutu. Tiedostoa luetaan vain osa kerrallaan. Joka lukukerralla tarkastetaan, mihin viimeksi jäätiin ja tallennetaan luettu kohta, jotta tiedostoa luetaan oikeasta kohdasta.

5.6 Äänen virtautus

Pakkaamaton ääni vie usein paljon tilaa. Tämän takia esimerkiksi kun halutaan toistaa musiikkia, ei ole järkevää ladata kokonaisen kappaleen verran pakkaamatonta ääntä koneen keskusmuistiin, vaan kappale on huomattavasti tehokkaampaa toistaa osissa, jolloin luetaan tietty määrä kappaletta muistiin, ja kun osa siitä on toistettu, niin luetaan aikaisempiin muistipaikkoihin lisää. Tätä toistomenetelmää kutsutaan äänen virtauttamiseksi (engl. streaming).

Virtauttamiseen käytetään rengaspuskuria. Rengaspuskuri on tietorakenne, joka koostuu tietyn kokoisesta puskurista. Tämän puskurin kummankin pään ajatellaan olevan kiinni toisissaan, jolloin kun sitä luettaessa saavutaan loppuun, niin jatketaankin lukemista alusta uudelleen. Rengaspuskuriin kirjoitetaan ja siitä luetaan samanaikaisesti. [24]

Kuvassa 7 on esimerkki tällaisesta rengaspuskurista. Kuvan puskurissa on kolmeosainen rengaspuskuri, jonka ensimmäistä osaa luetaan ja viimeiseen osaan kirjoitetaan. Nuolet kuvassa näyttävät miten kirjoitus- ja lukukohdat vaihtuvat ja mitä tapahtuu kun ollaan puskurin reunalla. Puskurin koko tässä esimerkissä on yhteensä 12288 tavua.



Kuva 7. Rengaspuskurin toiminta.

5.6.1 Äänidatanlukusäie

Koska äänen virtauttaminen vaatii jatkuvaa tiedoston lukemista sekä muistin käyttöä tarvitaan tätä toimintoa varten oma säie äänimoottoriin. Säikeen toteuttaminen äänenvirtautukselle varmistaa sen, että muun ohjelman suoritus ei pysähdy siksi aikaa kun ääntä toistetaan. Äänidatanlukusäie toteutetaan tässä äänimoottorissa siten, että se käynnistetään virtautettaville äänille siinä kohtaa kun ääntä aletaan toistaa. Kun ääni on toistettu loppuun asti, säie lopettaa toimintansa ja poistuu ohjelmasta. Jos ääni toistetaan uudestaa, niin sille luodaan jälleen uusi säie datan lukemista varten.

Säikeen synkronointi toteutetaan tapahtumilla (engl. event). Tällaisia tapahtumia voidaan jäädä säikeessä odottamaan, jolloin ei tarvita resursseja kuluttavaa kiertokyselyä (engl. polling). Tapahtumalla on käytännössä vain kaksi tilaa, jotka ovat asetettu tai ei asetettu. Asetettu tilaa voidaan jäädä säikeessä odottamaan joko tietyksi ajaksi tai niin pitkään kunnes tila vaihtuu. [25]

5.6.2 XAudio2:n takaisinkutsufunktiot

Kuten edellä käsiteltiin, tarvitsee äänen virtauttamista varten lukea äänitiedostoa pala kerrallaan. Tämä kuitenkin on vaikeaa, jos ei tiedetä milloin uusi puskuri on syytä syöttää lähdeäänelle. Tämä ongelma voidaan ratkaista käyttämällä XAudio2:n lähdeäänelle määriteltyjä takaisinkutsufunktioita.

XAudio2 osaa kutsua asiakasohjelman määrittelemiä funktioita asynkronisesti, kun tietty tapahtuma sattuu äänen prosessointi säikeessä. Tämä toteutetaan käyttämällä takaisinkutsufunktioita. Nämä funktiot määritellään johonkin tiettyyn rakenteeseen, joka annetaan joko XAudio2:lle tai XAudio2:n lähdeäänelle riippuen siitä ovatko funktiot koko äänijärjestelmän laajuiseen käyttöön vai vain yhden lähdeäänen käyttöön. Takaisinkutsufunktioissa ei voi suorittaa mitään raskaita toimenpiteitä, koska äänijärjestelmä joutuu odottamaan funktion suorituksen ajan. Tämän takia takaisinkutsufunktioilla asetetaan vain tapahtumia, jolloin joku toinen säie suorittaa tarvittavat toiminnot. [26][27]

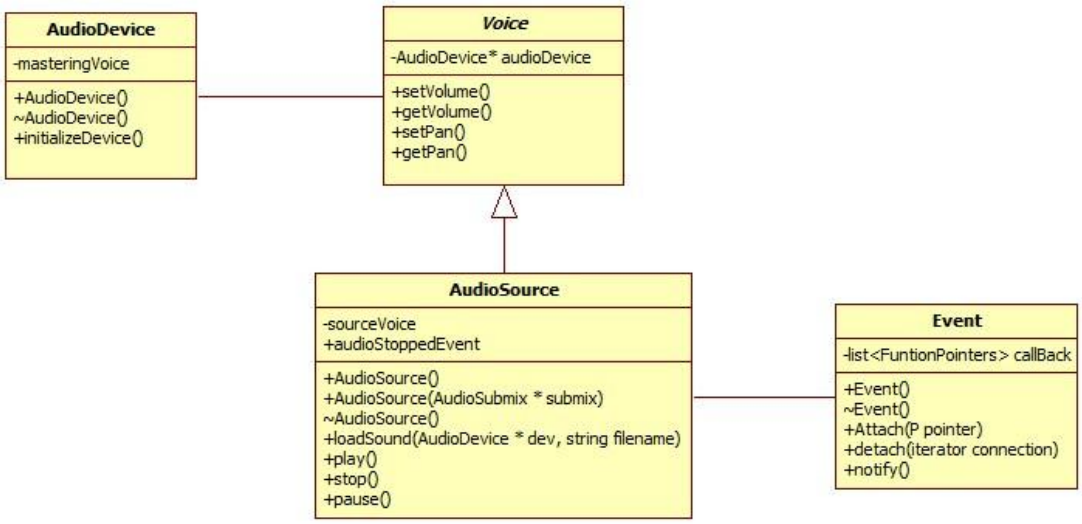
Äänimoottorin tiedoston lukeminen virtauttaessa toteutetaan siten, että luetaan äänidataa puskurin ensimmäiseen osaan ja syötetään se lähdeäänelle. Jos ääntä on tämän jälkeen jäljellä tiedostossa, luetaan se seuraavaan puskurin osaan, joka taas syötetään äänimoottorille. Tätä jatketaan kunnes äänidataa ei ole luettavaksi enää tai kunnes puskuri on täynnä, jolloin odotetaan takaisinkutsufunktiolta tapahtumaa, joka kertoo, että osa puskurista on toistettu jo. Kun tiedosto on luettu kokonaan, odotetaan takaisinkutsufunktioilta tapahtumia niin kauan, että puskureita ei enää ole lähdeäänellä. Tämän jälkeen voidaan puskuri, lähdeääni sekä säie siivota pois ohjelmasta.

5.7 Tarkkailijan toteutus

Asiakasohjelman sisällä on usein tarpeellista tietää milloin äänen toistaminen lopetetaan, koska silloin asiakasohjelman täytyy mahdollisesti suorittaa jonkinlaisia toimenpiteitä. Tämä on hyvin yksinkertaista äänimoottoriin toteuttaa käyttäen kiertokyselytekniikkaa, mutta se ei ole järkevää, koska kyseinen tekniikka saattaa olla hidas, jonka lisäksi se kuluttaa turhaan resursseja jatkuvaan kyselyyn. Tämän takia äänimoottoriin toteutetaan tarkkailija (engl. Observer).

Tarkkailijan avulla voidaan määritellä yhteys eri objektien välille siten, että kun tarkkailtava objekti muuttuu jotenkin, myös sitä tarkkailevat oliot muuttuvat. Tarkkailija vähentää tarvetta tehdä suoria riippuvuuksia luokkien välille, koska voidaan dynaamisesti lisätä tai poistaa tarkkailijoita. [28]

Kuvassa 8 on UML-kaavio, joka havainnollistaa miten tarkkailija on toteutettu äänimoottorissa. Tarkkailijaa tarvitaan toistaiseksi ainoastaan äänen loppumisen tarkkailuun, joten sitä varten on oma audioStoppedEvent-olio AudioSource-luokassa. Tämä tarkkailija-olio on käytännössä varasto ja rajapinta kaikille äänen loppumista tarkkaileville tarkkailijoille.



Kuva 8. Tarkkailija-objekti äänimoottorissa.

Tarkkailijaa käytetään äänimoottorissa siten, että kun ääni luodaan asiakas ohjelmassa, voidaan ääni-oliosta suoraan päästä käsiksi tarkkailija-olioon, jolloin sille voidaa attach-metodilla liittää jokin funktio tai jonkin olion metodi. Vastaavasti siitä voidaan poistaa funktioita kutsumalla detach-metodia.

Kun äänimoottorilla toistettava ääni loppuu, niin välittömästi kutsutaan äänimoottorin sisällä tarkkailija-olion notify-metodia. Tämä metodi käytännössä suorittaa kaikki tarkkailija-oliolle varastoidut funktiot ja metodit, jolloin kaikki tarkkailijat saavat tiedon äänentoistamisen loppumisesta. tämän toteuttaminen eroaa muistista toistettavan äänen sekä virtautettavan äänen välillä, koska niiden toteutustavat ovat hyvin erilaisia. Virtautettavan äänen säikeen lopussa voidaan helposti kutsua tarkkailijan notify-metodia, mutta muistista toistettavalla äänellä tällaista tilannetta ei ole. Muistista toistettavat äänet toistetaan täysin asynkronisesti, joten niiden tarkkailuun pitää toteuttaa oma säie. Tämä säie luodaan heti

kun ääni-olio luodaan, mutta säie jää odottamaan tapahtumaa, niin pitkään kunnes ääni ladataan virtauttamalla, jolloin säie tuhotaan tai kunnes ääntä on toistettu, jolloin säie suorittaa notify-metodin kutsumisen ja palaa takaisin odottamaan.

5.8 3D-äänien toteuttaminen

3D-äänien toteuttaminen äänimoottoriin hoidetaan XAudio2-kirjaston apukirjastolla X3DAudio. Tämä kirjasto on suunniteltu toteuttamaan 3D-äänten vaatimat laskutoimet ja ominaisuudet XAudio2:ssa. [11]

3D-ääniominaisuuksia käytetään äänissä, joilla voidaan ajatella olevan jonkinlaiset koordinaatit pelimaailmassa. Tämä tarkoittaa sitä, että esimerkiksi taustamusiikit ja vastaavat eivät käytä näitä ominaisuuksia, mutta ääniefektit ja mahdollisesti pelimaailmassa jostain kuuluvat musiikit käyttävät.

3D-äänien aikaan saamiseksi tarvitsemme kaksi hyvin olennaista käsitettä, Kuuntelija sekä äänenlähettäjän. Nämä kaksi käsitettä muodostavat kaiken tarpeellisen, mitä tarvitsemme 3D-äänien käsittelyyn.

Kuuntelija-objekti on usein lähes sama kuin 3D-maailmassa kamera. Tämä tarkoittaa usein sitä, että kuuntelija on samassa pisteessä kuin pelimaailmassa oleva pelihahmo. Kuuntelijalla on 3D-avaruudessa koordinaatit sekä suunta, joiden avulla voimme laskea miltä äänen kuuluisi kuulostaa kun se tulee jostain suunnasta.

Lähettäjä-objekti tarkoittaa äänilähdettä, jotain asiaa pelimaailmassa, josta tulee ääntä. Tämä voi olla esimerkiksi jokin pelimaailman hahmoista, jota pelaaja ei kontrolloi. Tälläkin objektilla on koordinaatit sekä suunta, mutta näiden lisäksi sille voidaan määritellä äänikartio. Äänikartion ominaisuudet määrittelevät sen miten ääni käyttäytyy kun se lähtee tästä objektista, esimerkiksi miten laajalle alueelle se kuuluu.

Kuuntelija- ja lähettäjäobjektit ovat molemmat X3Daudio-kirjastossa yksinkertaisia tietueita, mutta nämä useimmiten liitetään johonkin olioon. Äänimoottorissa

kuuntelija on liitetty osaksi äänilaitetta. Tämä johtuu siitä, että kuuntelijoita kussakin ohjelmassa voi olla ainoastaan yksi. Lähettäjä on äänimoottorissa liitetty ääniobjektiin, koska jokaisella äänellä todennäköisesti pitää olla omat koordinaattinsa sekä suuntansa.

Edellä mainittujen rakenteiden olemassaolo ei vielä riitä 3D-äänen toteutukseksi. Näitä rakenteita ennen pitää X3Daudio olla alustettu, ja niiden jälkeen tarvitsee laskea uudet määritykset äänille joka kerta kun kuuntelijan tai jonkun lähettäjän koordinaatit muuttuvat. Suositeltu laskentaväli äänten ominaisuuksille on joka 2. tai 3. kehys (engl. frame). [29]

5.9 Efektien lisääminen ääneen

Efektien lisääminen on todella kätevä lisäominaisuus äänimoottorilla, koska se vähentää tarvittavien äänitiedostojen määrää. Jokaista ympäristöä varten ei tarvitse olla omaa äänitiedostoa vaan yhtä ääntä voidaan muokata lisäämällä siihen jotain efektiä. Efekti tarkoittaa komponenttia, joka vastaanottaa äänidatan, muokkaa sitä ja lopuksi lähettää sen muokattuna eteenpäin.

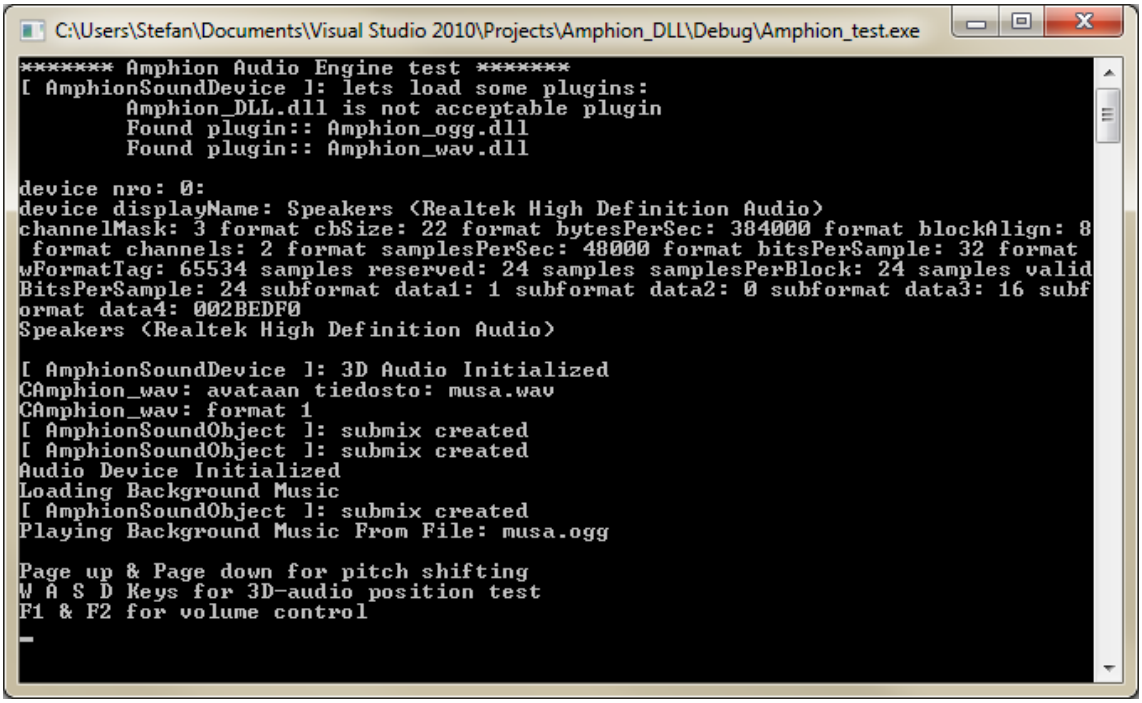
Mille tahansa XAudio2:n äänelle voidaan asettaa efektiketju (engl. effect chain). Efektiketjussa voi olla yksi tai useampi efekti. Kaikki siinä olevat efektit vaikuttavat jollain tavalla ääneen. Efektien vaatimat rakenteet ovat käytettävissä XAPOFX-kirjastossa, joka on Xaudio2:n apukirjasto efektien käsittelyä varten. [30]

Tämän opinnäytetyön äänimoottoriin toteutetaan vain tilakaiku-efekti. Tämä on todella tärkeä efekti, koska sillä saadaan ääneen lisättyä vaikutelma tilan koosta ja materiaalista, josta tilan seinät esimerkiksi koostuvat. Tämän efektin käsittely lisätään lähdeääntä hallinnoivaan luokkaan, jolloin jokaiselle yksittäiselle äänelle voidaan erikseen asettaa omanlainen tilakaikunsa. Käytännössä tilakaiku toteutetaan lisäämällä luokkaan metodi, joka ottaa parametriksi huoneen koon sekä seinien aihettaman diffuusion. Diffuusio määrittelee tilakaiulle seinien materiaalin tyypin. Esimerkiksi pienimmällä diffuusio-arvolla tarkoitetaan täysin sileää ja kovaa pintaa kun taas korkealla diffuusio-arvolla voidaan tarkoittaa pehmeää ja karvaista seinämateriaalia [31].

6 TESTAUS

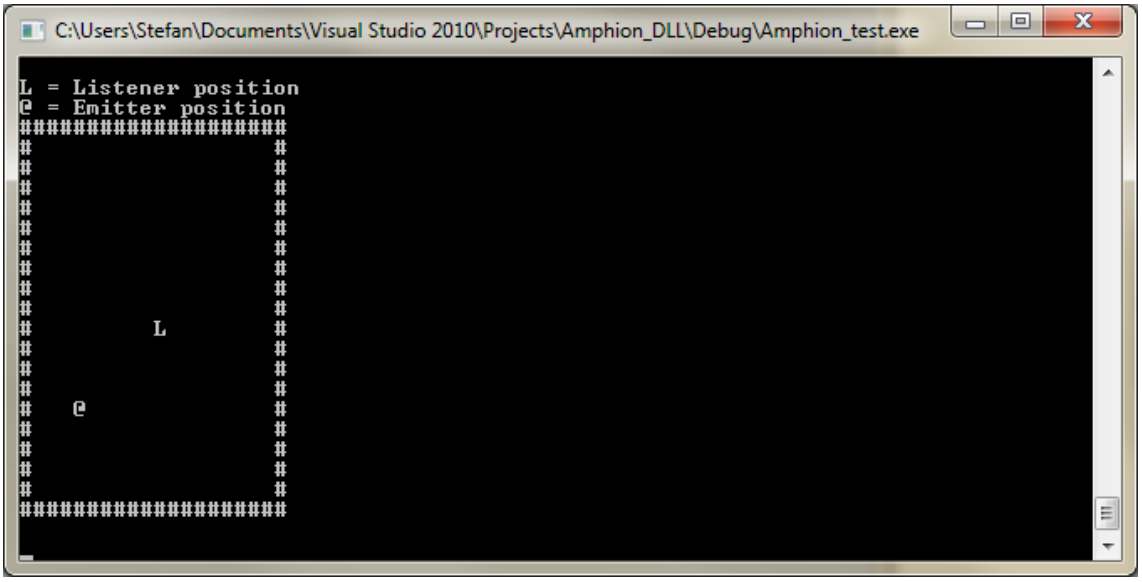
Äänimoottoria testattiin jatkuvasti sen kehittämisen yhteydessä. Tämä tarkoittaa sitä, että kun uusia ominaisuuksia saatiin valmiiksi, niitä myös testattiin heti, jolloin mahdolliset ongelmat tulevat mahdollisimman pian ilmi. Joissain tapauksissa uuden ominaisuuden lisääminen tai jonkun olemassa olevan ominaisuuden muuttaminen vaikuttaa myös muihin äänimoottorin osiin. Pienenkin muutoksen jälkeen on siis hyvä varmistaa, että äänimoottori on vielä vakaa ja toimii niin kuin sen pitää.

Kehitysvaiheen kaikki testaukset suoritettiin yksinkertaisella komentokehotteessa toimivalla asiakasohjelmalla. Tämä ohjelma loi sillan DLL-tiedostoon ja käytti DLL-tiedoston ominaisuuksia Äänimoottorin ohjelmointirajapinnan kautta. Kuvassa 9 on esimerkki testiohjelman suorittamisesta. Tässä esimerkissä Äänimoottori hakee ensin ohjelmistolisäkkeet, jotka ovat samassa kansiossa kuin suoritettava ohjelma. Tämän jälkeen tulostetaan äänilaitteen teknisiä tietoja. Lopuksi ladataan muutama äänitiedosto, joita aletaan ohjelmalla toistaa.



Kuva 9. Äänimoottorin testiohjelman suorittaminen.

3D-äänien toiminta oli myös tärkeä testata. Tätä varten komentokehotteessa toimivaan testiohjelmaan rakennettiin yksinkertainen toiminto, jolla nähdään lähettäjä- ja kuuntelija-objektien sijainti avaruudessa. Jotta testaus olisi mahdollisimman kattava, lisättiin siihen mahdollisuus liikuttaa lähettäjä-objektia. Tässä testiohjelmassa Lähettäjä-objektia liikuteltiin pienessä tilassa sekä vaaka- että pystysuuntaisesti. Kuvassa 10 on esimerkki tämän testin suorittamisesta.



Kuva 10. 3D-äänien testaaminen.

Äänimoottorin suorituskykyä testattiin myös lataamalla useita ääniä yhtäaikaisesti muistiin, jonka lisäksi toistettiin taustamusiikkia. Äänimoottori pystyi vaivatta toistamaan kymmeniä ääniä päällekkäin. Tämän lisäksi testattiin isokokoisen OGG Vorbis -tiedoston toistamista purkamalla se suoraan koneen keskusmuistiin. OGG Vorbis -tiedosto oli kooltaan 17,8 MB, jolloin muistiin purettuna se vei 207 MB. Testikoneella kappaleen muistiin purkaminen kesti keskimäärin 20 s., jonka jälkeen äänimoottori pääsi varsinaisesti toistamaan kappaletta. Äänimoottori pystyi samanaikaisesti toistamaan myös muita ääniä.

7 TULOKSET

Lopputuloksena tästä opinnäytetyöstä saatiin toimiva DirectX:n XAudio2-kirjastoon perustuva Amphion-niminen äänimoottori. Kaikki vaatimusmäärittelyssä esitetyt ominaisuudet toteutettiin onnistuneesti.

Toteutettu äänimoottori on käytännössä vain yksi DLL-tiedosto, jota muut ohjelmat voivat käyttää. Yksinään tämä DLL-tiedosto ei kuitenkaan riitä äänentoistamiseen vaan tarvitaan tuki erilaisille ääniformaateille. Tähän äänimoottoriin toteutettiin tuki kahdelle eri ääniformaatille, jotka ovat WAV ja OGG Vorbis. Ääniformaattien tuki toteutettiin ohjelmalisäkearkkitehtuurin mukaisesti, eli uuden ääniformaatin tuen lisääminen äänimoottorille on varsin yksinkertaista, koska itse äänimoottoriin ei tarvitse enää tehdä muutoksia. Jokaista ääniformaattia varten äänimoottoriin lisätään oma ohjelmistolisäke eli DLL-tiedosto. Äänimoottori havaitsee ohjelmistolisäkkeiksi sopivat tiedostot toimintahakemistossaan, jonka lisäksi sille voidaan käsin lisätä ohjelmistolisäkkeitä.

Äänimoottorin äänigraafi toteutettiin onnistuneesti siten, että uusia ääniä voidaan yksinkertaisesti luoda ohjelmassa. Ääniä voidaan myös toistaa päällekkäin, sekä niitä voidaan ryhmitellä halutulla tavalla. Tämä lisäksi äänigraafin lähdeääniosaan toteutettiin yksinkertainen tilakaiku. Tilakaiun päälle asettamisen yhteydessä sille annetaan parametrina huoneen koko, jonka mukaan se toteuttaa tilakaiun.

Valmiin äänimoottorin toimintaa myös testattiin kahdessa eri tietokoneessa ja useilla erilaisilla ohjelmilla. Äänimoottorilla toistettiin useita eri äänitiedostoja sekä muistiin ladattuina että virtautettuina. Muistiin ladattuja tiedostoja myös toistettiin kymmeniä kertoja samanaikaisesti päällekkäin. Testeissä havaittiin pieniä ohjelmavirheitä, jotka korjattiin välittömästi. Nämä testit olivat kuitenkin varsin pieniä, joten niissä ei kaikkia mahdollisia ongelmia testattu.

Äänimoottorin ohjelmointirajapinnasta saatiin toteutettua sellainen, että sitä on yksinkertaista käyttää. Normaalin taustamusiikin toistaminen ja muutaman ää-

nen toistaminen voidaan toteuttaa vain muutamalla rivillä koodia. Koodiesimerkissä 5 on toimiva esimerkki tätä toteutettua äänimoottoria käyttävästä ohjelmasta. Tämä ohjelma alustaa äänimoottorin, minkä jälkeen se lataa OGG Vorbis –tiedoston, joka sisältää musiikkia. Tämä tiedosto toistetaan virtauttamalla kunnes käyttäjä painaa jotain painiketta näppäimistöltä tai musiikki on toistettu kokonaan.

Koodiesimerkki 5. Toteutetun äänimoottorin käyttöesimerkki.

```
#include <Windows.h>
#include "Amphion_DLL.h"

using namespace Amphion;

int main()
{
    ASoundDevice dev;
    ASoundObject musa;
    if (dev.initializeDevice())
    {
         if (!musa.loadSound(&dev,"musa.ogg",true))
            MessageBoxA(0,"Tiedostoa ei löydy","Amphion",0);

         musa.play();
    }
    system("pause");
    return 0;
}
```

8 YHTEENVETO

Äänimoottorin suunnittelu ja toteutus on monipuolinen projekti, jossa täytyy tutustua digitaalisen äänen rakenteeseen, erilaisiin äänikirjastoihin ja äänimoottoreihin sekä erilaisiin ääniformaatteihin. Projektin suunnittelu ja toteutus vaativat paljon tutkimista etenkin sen takia, että tässä projektissa käytetyn XAudio2-kirjaston käytöstä oli melko vähän esimerkkejä ja materiaalia.

Työssä piti tutustua ääniformaattien ja –moottoreiden lisäksi myös erilaisiin ohjelmointimenetelmiin. Monet vastaan tulleista ohjelmointimenetelmistä olivat täysin uusia minulle, mikä teki äänimoottorin toteuttamisesta ohjelmointitaitojen kannalta myös hyvin opettavaisen projektin. Ohjelmalisäkearkkitehtuurin toteuttaminen eri ääniformaatteja varten sekä tarkkailijan toteuttaminen olivat minulle täysin uusia asioita. Näiden ohjelmointimenetelmien lisäksi WAV- ja OGG Vorbis –tiedostojen lukeminen oli minulle uusi asia.

Äänimoottorin joitain osia voidaan kehittää vielä huomattavasti paremmiksi. Tästä esimerkkinä on efektien lisääminen ääneen, joka olisi mahdollista toteuttaa huomattavasti modulaarisemmin. Yksi idea paremmasta toteutustavasta olisi luoda efektejä varten kokonaan oma luokka. Tästä luokasta saisi kehitettyä ohjelman ajon aikanakin uusia efektejä, joita voisi liittää toistettavaan ääneen.

Äänimoottorin toteutus onnistui hyvin, ja äänimoottori onkin siirretty jatkokehitykseen FantasyCraftin meneillään olevaan peliprojektiin.

LÄHTEET

[1] Meigs, T., Audio, Ultimate Game Design s. 74

[2] Äänipää, digitaalinen ääni, [www-dokumentti], saatavilla: http://www.aanipaa.tamk.fi/digi_1.htm (Luettu: 19.2.2012)

[3] Wikipedia, Pulse-code modulation, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/Pulse-code_modulation (luettu: 19.2.2012)

[4] Sourceforge, Audiere [www-dokumentti], saatavilla: audiere.sourceforge.net WWW-SIVU, (luettu 21.1.2012)

[5] WIkipedia, OpenAL, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/OpenAL (luettu: 21.1.2012)

[6] Un4seen developements, BASS Audio library, [www-dokumentti], saatavilla: http://www.un4seen.com (luettu 22.4.2012)

[7] FMOD Ex, [www-dokumentti], saatavilla: http://www.fmod.org/wiki/index.php5?title=API_-_Introduction (luettu 1.4.2012)

[8] Wikipedia, XACT, [www-dokumentti] http://en.wikipedia.org/wiki/Cross-platform_Audio_Creation_Tool (luettu 22.4.2012)

[9] Wikipedia, Dynamic Link Library, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/Dynamic-link_library (luettu 4.2.2012)

[10] Microsoft, XAudio2, [www-dokumentti], saatavilla http://msdn.microsoft.com/en-us/library/windows/desktop/ee415813%28v=vs.85%29.aspx (luettu: 4.2.2012)

[11] Microsoft, X3DAudio, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415717%28v=vs.85%29.aspx (luettu: 22.3.2012)

[12] Wikipedia, WAV, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/WAV (luettu: 24.3.2012)

[13] Audio File Format Specifications, [www-dokumentti], saatavilla: http://www-mmsp.ece.mcgill.ca/documents/audioformats/wave/wave.html (luettu: 24.3.2012)

[14] Wikipedia, OGG, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/Ogg (luettu: 24.3.2012)

[15] Wikipedia, OGG page, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/Ogg_page (luettu: 15.4.2012)

[16] Wikipedia, Vorbis, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/Vorbis (luettu: 15.4.2012)

[17] Microsoft, Audio graph, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415741%28v=vs.85%29.aspx (luettu: 8.4.2012)

[18] Microsoft, XAudio2 Voices, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415825%28v=vs.85%29.aspx (luettu: 8.4.2012)

[19] Microsoft, WAVEFORMAT, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/dd757712%28v=vs.85%29.aspx (luettu: 8.4.2012)

[20] Microsoft, WAVEFORMATEX, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/dd390970%28v=vs.85%29.aspx (luettu: 8.4.2012)

[21] Microsoft, XAUDIO2 BUFFER, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer%28v=vs.85%29.aspx (luettu: 8.4.2012)

[22] Cygon's Blog, Plugin architecture, [www-dokumentti], saatavilla: http://blog.nuclex-games.com/tutorials/cxx/plugin-architecture/ (luettu: 23.3.2012)

[23] Prata, S., Tiedostojen I/O –toiminnot, C++ ohjelmointi s. 500 - 507

[24] Wikipedia ,Circular buffer, [www-dokumentti], saatavilla: http://en.wikipedia.org/wiki/Circular_buffer (luettu: 15.4.2012)

[25] Codeproject, Thread synchronization for beginners, [www-dokumentti], saatavilla: http://www.codeproject.com/Articles/7953/Thread-Synchronization-for-Beginners (luettu: 15.4.2012)

[26] Microsoft, XAudio2 callbacks, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415745%28v=vs.85%29.aspx (luettu: 8.4.2012)

[27] Microsoft, XAudio2 source callbacks, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415769%28v=vs.85%29.aspx (luettu: 8.4.2012)

[28] Gamma, E., Helm, R., Johnson, R., Vlissides, J.*, Observer*, Design Patterns: Elements of Reusable Object-Oriented Software, 1995, s. 326 - 337

[29] Microsoft, How to: Integrate X3DAudio with XAudio2, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415798%28v=vs.85%29.aspx (luettu: 1.4.2012)

[30] Microsoft, XAudio2 Audio Effects, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415756(v=vs.85).aspx (luettu: 5.5.2012)

[31] Microsoft, FXREVERB_PARAMETERS structure, [www-dokumentti], saatavilla: http://msdn.microsoft.com/en-us/library/windows/desktop/microsoft.directx_sdk.xapofx.fxreverb_parameters(v=vs.85).aspx (luettu: 5.5.2012)